

Gestione dei processi

- 2.1** Definizione di processo
- 2.2** Stati di un processo
- 2.3** Rappresentazione dei processi
- 2.4** Cambio di contesto
- 2.5** Scheduling
- 2.6** Operazioni sui processi
- 2.7** Processi e thread

2.1 Definizione di processo

- Informalmente, il termine processo viene utilizzato per indicare un programma in esecuzione.
- *“Processo: attività controllata da un programma che si svolge su un processore”*
- È **l'unità di esecuzione** all'interno del S.O.
 - processi sequenziali
 - un S.O. multiprogrammato consente l'esecuzione concorrente di più processi
 - NOTA: se esiste distinzione tra **processi pesanti** (*processi propriamente detti*) e **processi leggeri** (*thread*), le precedenti definizioni valgono per i processi leggeri.

Confronto tra esecuzione sequenziale ed esecuzione multi-tasking

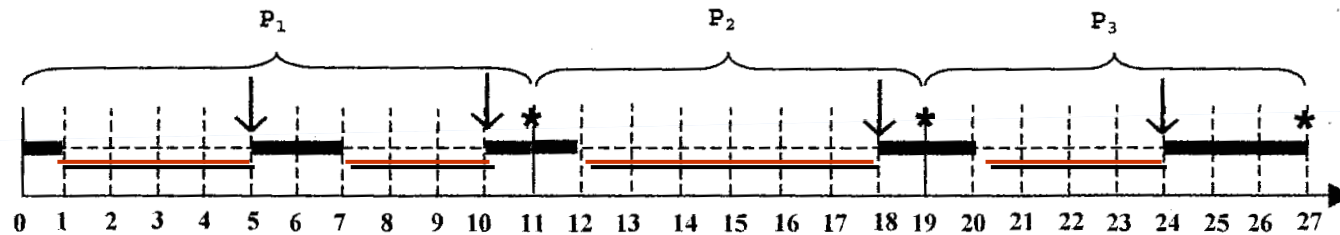


Figura 1.4 Esecuzione sequenziale.

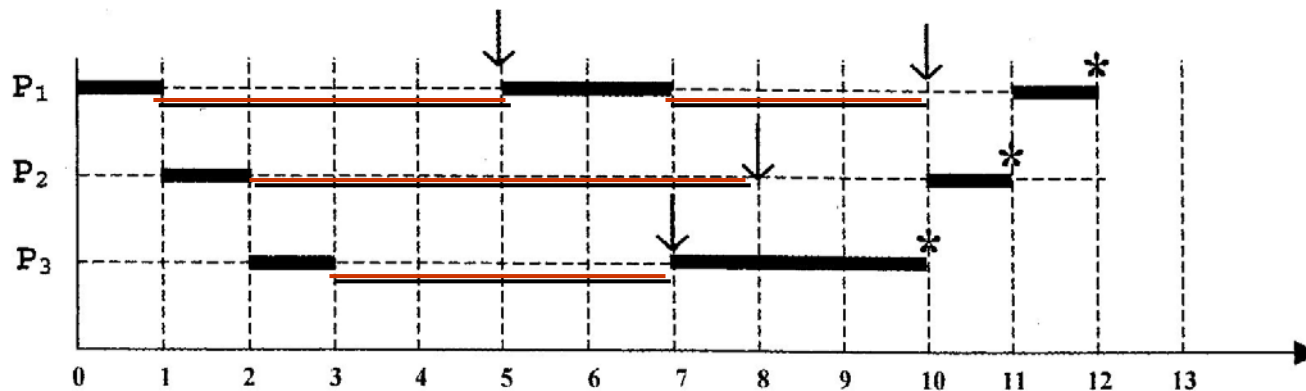


Figura 1.5 Esecuzione in multi-tasking.

Programma e processo

- **Programma:** sequenza di operazioni da eseguire (*flusso*)
- **Processo:** attività di esecuzione, generalmente discontinua

==> A uno stesso programma possono corrispondere più processi, che ciascuno rappresenta l'esecuzione dello stesso codice in tempi diversi e/o con dati diversi

Rappresentazione del processo

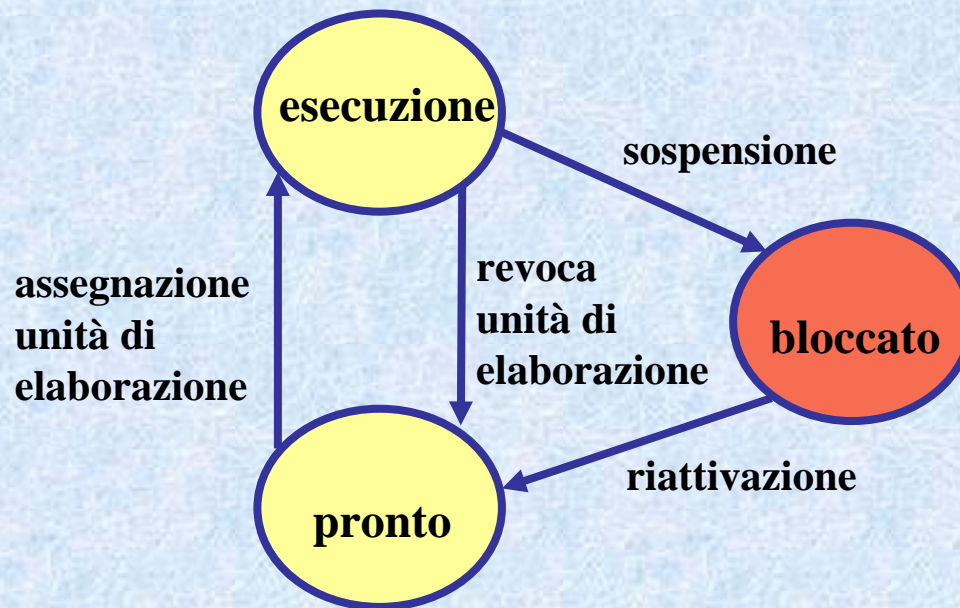
- Un processo è rappresentato da:
 - Codice
 - Dati
 - Stack
 - Contesto (Program Counter, Registri)
 - **Stato**
- Ad un processo sono associate delle risorse:
 - Memoria
 - Archivi aperti
 - Dispositivi di I/O

2.2 Stati di un processo



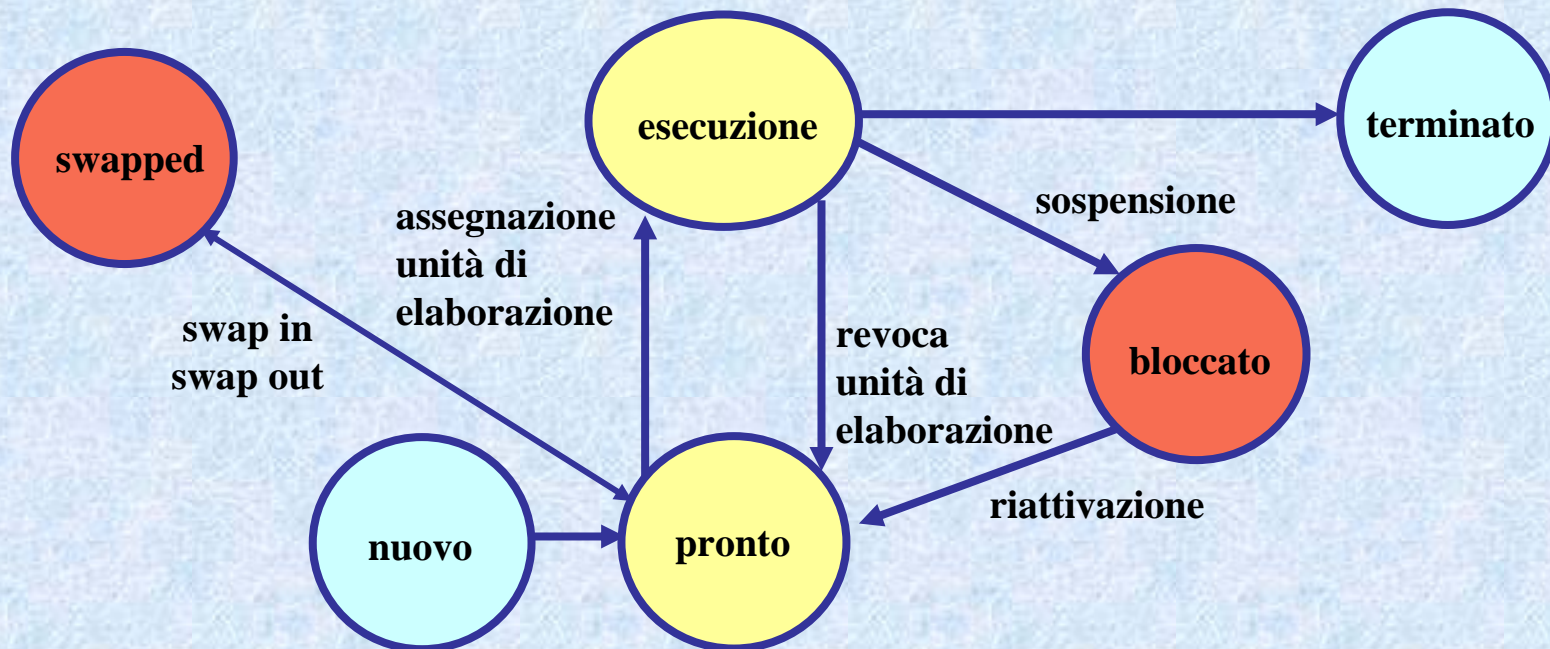
Questa situazione si verifica solo se per ogni processo è sempre disponibile un processore

- Se il numero di processori è minore del numero dei processi (esempio: sistema monoprocesso multiprogrammato), lo stato attivo si sdoppia in:
 - **pronto** (in attesa di una CPU)
 - **in esecuzione** (utilizza la CPU)



Altri stati:

- **nuovo** (creazione di un nuovo processo)
- **terminato** (terminazione del processo)
- **swapped** (scaricato in memoria secondaria, e quindi non schedabile)



2.3 Rappresentazione dei processi

- **Descrittore del processo ovvero Process Control Block (PCB):** struttura dati associata ad ogni processo.
- **Tabella dei processi:** contiene i descrittori di tutti i processi

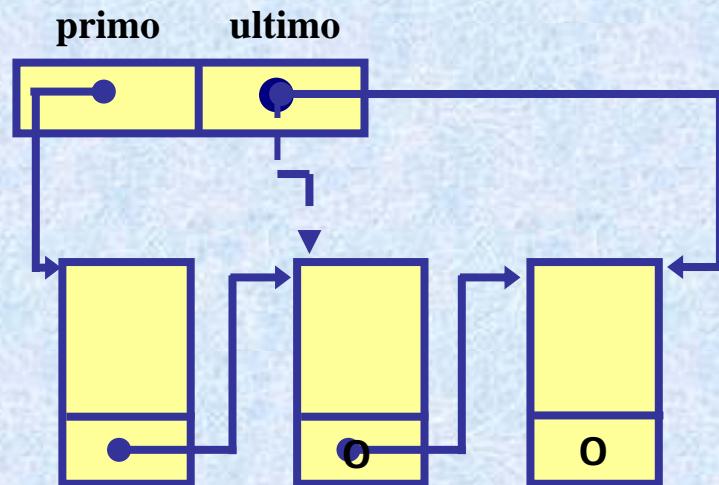
Descrittore di un processo

- **Nome del processo**
generalmente è l'indice del descrittore nella Tabella dei Processi)
- **Stato del processo**
- **Contesto del processo**
immagine dei registri generali e speciali
- **Modalità di servizio dei processi**
priorità, parametri di scheduling
eventuali scadenze temporali (nei sistemi in tempo reale)
- **Risorse di memoria assegnate**
- **Altre risorse assegnate**
archivi aperti ...
- **Puntatore (a descrittore di processo)**

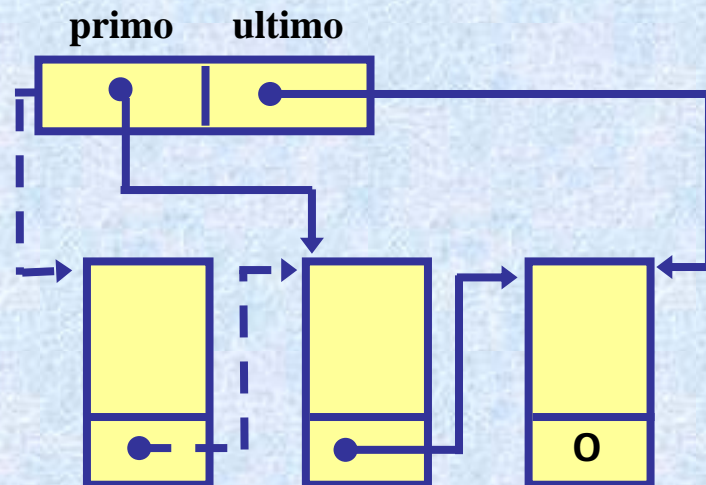
Code di processi

- Coda di processi pronti
- Code di processi bloccati

a) inserzione nella coda



b) rimozione dalla coda



- Registro del processo in esecuzione

2.4 Cambio di contesto

Avviene quando l'uso della CPU viene commutato da un processo ad un altro

==> Commutazione sempre innescata da un'interruzione (SVC o di altro tipo)

- **Salvataggio:** il contesto del processo in esecuzione è salvato nel suo descrittore
- Il descrittore è inserito nella coda dei processi bloccati o pronti
- **(short term) scheduling:** viene selezionato un processo pronto; il suo descrittore viene rimosso dalla coda dei processi pronti e viene puntato dal registro *In Esecuzione*
- **Ripristino:** il contesto del processo *In Esecuzione* viene caricato nei registri del processore

Cambio di contesto

Overhead

- Salvataggio e ripristino dei registri
- Inserimento del descrittore in una coda; selezione ed estrazione del descrittore del processo da mandare in esecuzione (short-term scheduling)
- Invalidazione e ricostituzione della cache
- Operazioni indotte sul gestore della memoria (eventuale swap-out e swap-in, picco di eccezioni di indirizzo (*cache della MMU*) e page-faults nei sistemi con paginazione)

Memoria Cache

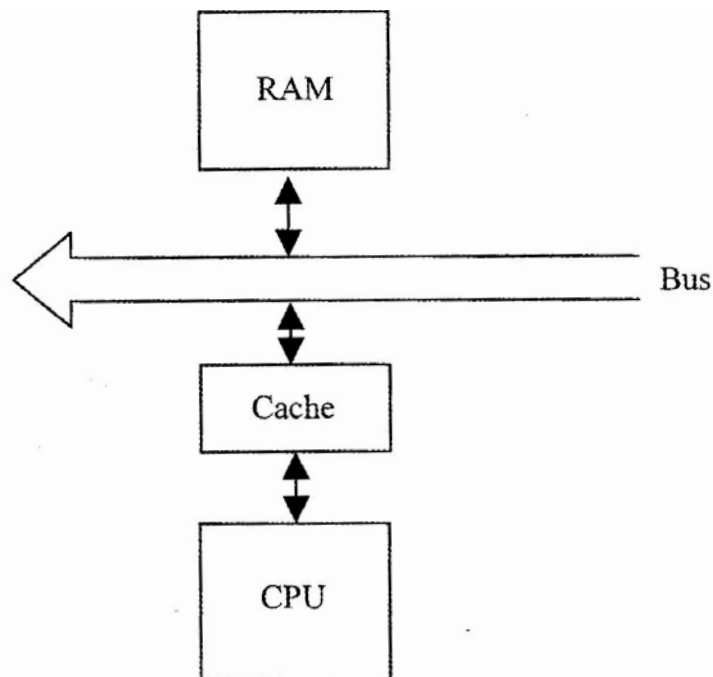


Figura 1.8 La memoria cache.

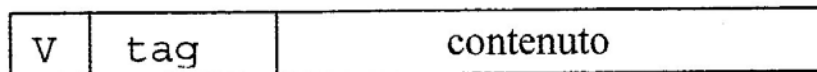


Figura 1.9 Elemento della cache.

Cambio di contesto: esempio (1)

Si considera un processore che dispone dei registri speciali PC, PS e SP e dei registri generali R1 e R2 (in un'unica copia);

Il sistema riserva in memoria un'area per il vettore di interruzione e per lo stack del nucleo ;

Il processore ha il seguente comportamento:

- al riconoscimento di un'interruzione, l'hardware instaura lo stato supervisore e disabilita le interruzioni, salva i registri speciali (compreso PC) nello stack del nucleo e salta all'indirizzo specificato dal vettore di interruzione (punto di ingresso della funzione di servizio).
- l'istruzione IRET ripristina i registri speciali (compreso PC) dallo stack del nucleo e contemporaneamente instaura lo stato utente e riabilita le interruzioni. Pertanto l'istruzione successiva è quella il cui indirizzo è stato ripristinato in PC.

Spetta alla funzione di servizio:

- Prima di ogni altra operazione, salvare i registri generali. Si conviene che anche questi registri vengano salvati nello stack del nucleo.
- Terminata l'esecuzione del corpo della funzione di servizio, ripristinare i registri generali, prelevando i rispettivi valori dallo stack del nucleo, e quindi eseguire l'istruzione IRET.

Ipotesi A): il processo P1 esegue l'istruzione SVC per invocare una primitiva, al termine della quale P1 rimane in esecuzione

1) situazione iniziale, durante l'esecuzione dell'istruzione SVC (STATO UTENTE)

Descrittore di P1	
Stato	Esecuzione
PC	????
PS	16F2
SP	????
R1	????
R2	????

Descrittore di P2	
Stato	Pronto
PC	A12C
PS	16F2
SP	A275
R1	25CC
R2	F012

Stack del nucleo	
0FFF	
1000	
1001	
1002	
1003	
1004	

Registri	
PC	1880
PS	16F2
SP	2880
R1	4500
R2	CD31

INDIRIZZO	5000
PAROLA DI STATO	0045
Vettore di interruzione	

Origine dello stack del nucleo	0FFF
--------------------------------	------

Cambio di contesto: esempio (2)

Ipotesi A): il processo P1 esegue l'istruzione SVC per invocare una primitiva, al termine della quale P1 rimane in esecuzione

1) situazione iniziale, durante l'esecuzione dell'istruzione SVC (STATO UTENTE)

Descrittore di P1	
Stato	Esecuzione
PC	????
PS	16F2
SP	????
R1	????
R2	????

Descrittore di P2	
Stato	Pronto
PC	A12C
PS	16F2
SP	A275
R1	25CC
R2	F012

Stack del nucleo	
0FFF	
1000	
1001	
1002	
1003	
1004	

Registri	
PC	1880
PS	16F2
SP	2880
R1	4500
R2	CD31

INDIRIZZO	5000
PAROLA DI STATO	0045
Vettore di interruzione	

Origine dello stack del nucleo	0FFF
--------------------------------	------

2) dopo il riconoscimento dell'interruzione (STATO SUPERVISORE)

Descrittore di P1	
Stato	Esecuzione
PC	????
PS	16F2
SP	????
R1	????
R2	????

Descrittore di P2	
Stato	Pronto
PC	A12C
PS	16F2
SP	A275
R1	25CC
R2	F012

Stack del nucleo	
0FFF	
1000	1880
1001	16F2
1002	2880
1003	
1004	

Registri	
PC	5000
PS	0045
SP	1002
R1	4500
R2	CD31

INDIRIZZO	5000
PAROLA DI STATO	0045
Vettore di interruzione	

Origine dello stack del nucleo	0FFF
--------------------------------	------

Cambio di contesto: esempio (3)

Ipotesi A): il processo P1 esegue l'istruzione SVC per invocare una primitiva, al termine della quale P1 rimane in esecuzione

2) dopo il riconoscimento dell'interruzione (STATO SUPERVISORE)

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
Stato	Esecuzione	Stato	Pronto	0FFF		PC	5000
PC	????	PC	A12C	1000	1880	PS	0045
PS	16F2	PS	16F2	1001	16F2	SP	1002
SP	????	SP	A275	1002	2880	R1	4500
R1	????	R1	25CC	1003		R2	CD31
R2	????	R2	F012	1004			

3) dopo il salvataggio temporaneo dei registri generali nello stack del nucleo (STATO SUPERVISORE)

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
Stato	Esecuzione	Stato	Pronto	0FFF		PC	5000+ ??
PC	????	PC	A12C	1000	1880	PS	0045
PS	16F2	PS	16F2	1001	16F2	SP	1004
SP	????	SP	A275	1002	2880	R1	????
R1	????	R1	25CC	1003	4500	R2	????
R2	????	R2	F012	1004	CD31		

Cambio di contesto: esempio (4)

Ipotesi A): il processo P1 esegue l'istruzione SVC per invocare una primitiva, al termine della quale P1 rimane in esecuzione

3) quando termina il corpo della procedura di gestione (STATO SUPERVISORE)

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
Stato	Esecuzione	Stato	Pronto	0FFF		PC	5000+ ??
PC	????	PC	A12C	1000	1880	PS	0045
PS	16F2	PS	16F2	1001	16F2	SP	1004
SP	????	SP	A275	1002	2880	R1	????
R1	????	R1	25CC	1003	4500	R2	????
R2	????	R2	F012	1004	CD31		

4) durante l'istruzione IRET (indirizzo 5100), dopo il ripristino dei registri generali (STATO SUPERVISORE)

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
Stato	Esecuzione	Stato	Pronto	0FFF		PC	5100
PC	????	PC	A12C	1000	1880	PS	0045
PS	16F2	PS	16F2	1001	16F2	SP	1002
SP	????	SP	A275	1002	2880	R1	4500
R1	????	R1	25CC	1003		R2	CD31
R2	????	R2	F012	1004			

Cambio di contesto: esempio (5)

Ipotesi A): il processo P1 esegue l'istruzione SVC per invocare una primitiva, al termine della quale P1 rimane in esecuzione

4) durante l'estrazione dell'istruzione IRET (indirizzo 5100), dopo il ripristino dei registri generali (STATO SUPERVISORE)

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
Stato	Esecuzione	Stato	Pronto	0FFF		PC	5100
PC	????	PC	A12C	1000	1880	PS	0045
PS	16F2	PS	16F2	1001	16F2	SP	1002
SP	????	SP	A275	1002	2880	R1	4500
R1	????	R1	25CC	1003		R2	CD31
R2	????	R2	F012	1004			

5) al termine dell'esecuzione dell'istruzione IRET (STATO UTENTE)

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
Stato	Esecuzione	Stato	Pronto	0FFF		PC	1880
PC	????	PC	A12C	1000		PS	16F2
PS	16F2	PS	16F2	1001		SP	2880
SP	????	SP	A275	1002		R1	4500
R1	????	R1	25CC	1003		R2	CD31
R2	????	R2	F012	1004			

INDIRIZZO	5000
PAROLA DI STATO	0045
Vettore di interruzione	

Origine dello stack del nucleo	0FFF
--------------------------------	------

Cambio di contesto: esempio (6)

Ipotesi B): Il processo P1 esegue l'istruzione SVC per invocare una primitiva, che determina il passaggio in esecuzione del processo pronto P2, e la sospensione di P1

1) situazione iniziale, durante l'esecuzione dell'istruzione SVC (STATO UTENTE)

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
Stato	Esecuzione	Stato	Pronto	0FFF		PC	1880
PC	????	PC	A12C	1000		PS	16F2
PS	16F2	PS	16F2	1001		SP	2880
SP	????	SP	A275	1002		R1	4500
R1	????	R1	25CC	1003		R2	CD31
R2	????	R2	F012	1004			

INDIRIZZO	5000
PAROLA DI STATO	0045
Vettore di interruzione	

Origine dello stack del nucleo	0FFF
--------------------------------	------

2) dopo il riconoscimento dell'interruzione (STATO SUPERVISORE)

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
Stato	Esecuzione	Stato	Pronto	0FFF		PC	5000
PC	????	PC	A12C	1000	1880	PS	0045
PS	16F2	PS	16F2	1001	16F2	SP	1002
SP	????	SP	A275	1002	2880	R1	4500
R1	????	R1	25CC	1003		R2	CD31
R2	????	R2	F012	1004			

INDIRIZZO	5000
PAROLA DI STATO	0045
Vettore di interruzione	

Origine dello stack del nucleo	0FFF
--------------------------------	------

Cambio di contesto: esempio (7)

Ipotesi B): Il processo P1 esegue l'istruzione SVC per invocare una primitiva, che determina il passaggio in esecuzione del processo pronto P2

2) dopo il riconoscimento dell'interruzione (STATO SUPERVISORE)

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
Stato	Esecuzione	Stato	Pronto	0FFF		PC	5000
PC	????	PC	A12C	1000	1880	PS	0045
PS	16F2	PS	16F2	1001	16F2	SP	1002
SP	????	SP	A275	1002	2880	R1	4500
R1	????	R1	25CC	1003		R2	CD31
R2	????	R2	F012	1004			

3) dopo il salvataggio temporaneo dei registri generali nello stack del nucleo (STATO SUPERVISORE)

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
Stato	Esecuzione	Stato	Pronto	0FFF		PC	5000+ ??
PC	????	PC	A12C	1000	1880	PS	0045
PS	16F2	PS	16F2	1001	16F2	SP	1004
SP	????	SP	A275	1002	2880	R1	????
R1	????	R1	25CC	1003	4500	R2	????
R2	????	R2	F012	1004	CD31		

Cambio di contesto: esempio (8)

Ipotesi B): Il processo P1 esegue l'istruzione SVC per invocare una primitiva, che determina il passaggio in esecuzione del processo pronto P2

3) dopo il salvataggio temporaneo dei registri generali nello stack del nucleo (STATO SUPERVISORE)

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
Stato	Esecuzione	Stato	Pronto	0FFF		PC	5000+ ??
PC	????	PC	A12C	1000	1880	PS	0045
PS	16F2	PS	16F2	1001	16F2	SP	1004
SP	????	SP	A275	1002	2880	R1	????
R1	????	R1	25CC	1003	4500	R2	????
R2	????	R2	F012	1004	CD31		

4) dopo il salvataggio del contesto del processo P1 e il caricamento del contesto del processo P2 nello stack del nucleo (STATO SUPERVISORE)

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
Stato	Bloccato	Stato	Esecuzione	0FFF		PC	5000+ ??
PC	1880	PC	A12C	1000	A12C	PS	0045
PS	16F2	PS	16F2	1001	16F2	SP	1004
SP	2880	SP	A275	1002	A275	R1	????
R1	4500	R1	25CC	1003	25CC	R2	????
R2	CD31	R2	F012	1004	F012		

Cambio di contesto: esempio (9)

Ipotesi B): Il processo P1 esegue l'istruzione SVC per invocare una primitiva, che determina il passaggio in esecuzione del processo pronto P2

4) dopo il salvataggio del contesto del processo P1 e il caricamento del contesto del processo P2 nello stack del nucleo (STATO SUPERVISORE)

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
Stato	Bloccato	Stato	Esecuzione	0FFF		PC	5100+ ??
PC	1880	PC	A12C	1000	A12C	PS	0045
PS	16F2	PS	16F2	1001	16F2	SP	1004
SP	2880	SP	A275	1002	A275	R1	????
R1	4500	R1	25CC	1003	25CC	R2	????
R2	CD31	R2	F012	1004	F012		

5) durante l'istruzione IRET (indirizzo 5100), immediatamente successive al ripristino dei registri generali (STATO SUPERVISORE)

Descrittore di P1		Descrittore di P2		Stack del nucleo		Registri	
Stato	Bloccato	Stato	Esecuzione	0FFF		PC	5100
PC	1880	PC	A12C	1000	A12C	PS	0045
PS	16F2	PS	16F2	1001	16F2	SP	1002
SP	2880	SP	A275	1002	A275	R1	25CC
R1	4500	R1	25CC	1003		R2	F012
R2	CD31	R2	F012	1004			

Cambio di contesto: esempio (10)

Ipotesi B): Il processo P1 esegue l'istruzione SVC per invocare una primitiva, che determina il passaggio in esecuzione del processo pronto P2

5) durante l'estrazione dell'istruzione IRET (indirizzo 5100), immediatamente successive al ripristino dei registri generali (STATO SUPERVISORE)

Descrittore di P1	
Stato	Bloccato
PC	1880
PS	16F2
SP	2880
R1	4500
R2	CD31

Descrittore di P2	
Stato	Esecuzione
PC	A12C
PS	16F2
SP	A275
R1	25CC
R2	F012

Stack del nucleo	
0FFF	
1000	A12C
1001	16F2
1002	A275
1003	
1004	

Registri	
PC	5100
PS	0045
SP	1002
R1	25CC
R2	F012

6) al termine dell'esecuzione dell'istruzione IRET (STATO UTENTE)

Descrittore di P1	
Stato	Bloccato
PC	1880
PS	16F2
SP	2880
R1	4500
R2	CD31

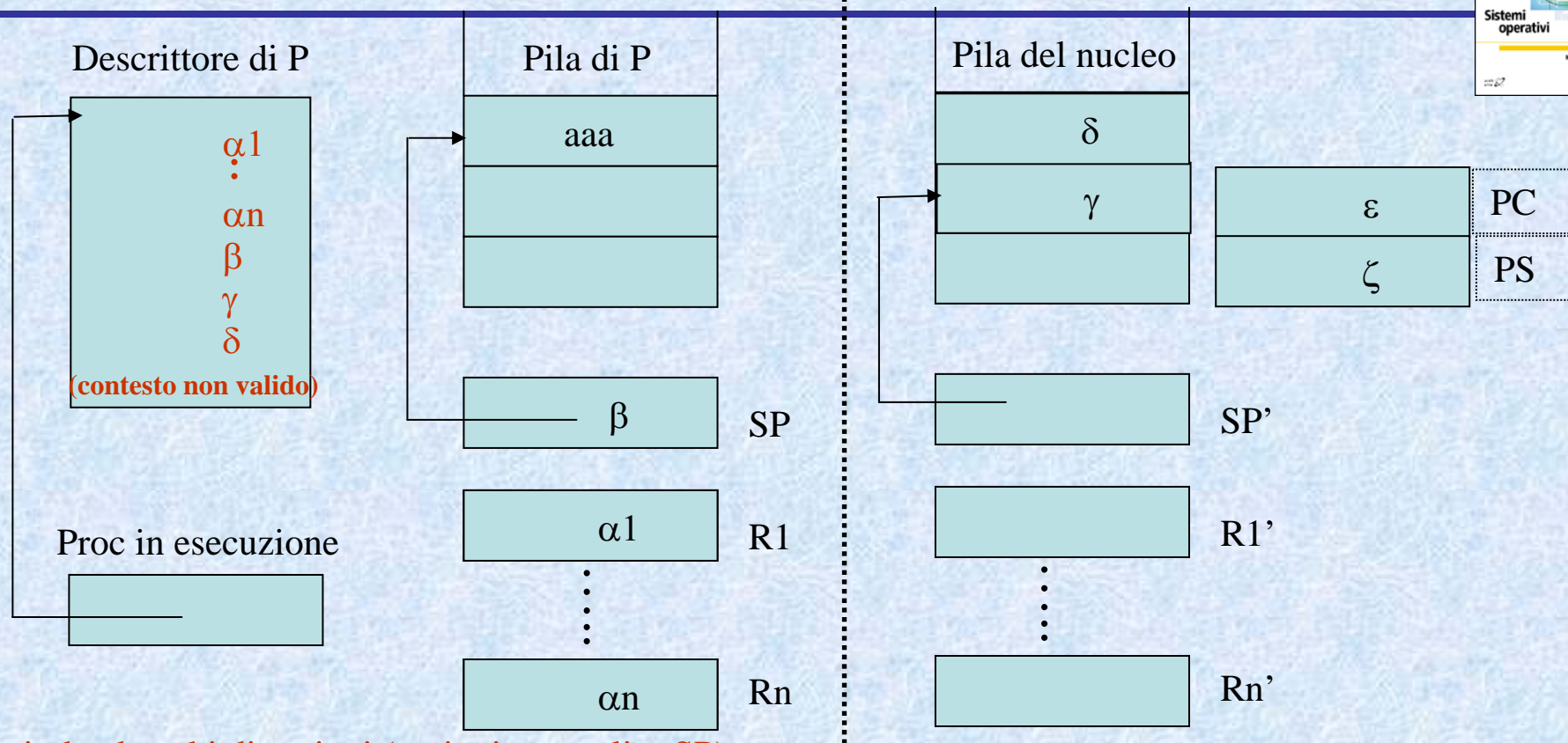
Descrittore di P2	
Stato	Esecuzione
PC	A12C
PS	16F2
SP	A275
R1	25CC
R2	F012

Stack del nucleo	
0FFF	
1000	
1001	
1002	
1003	
1004	

Registri	
PC	A12C
PS	16F2
SP	A275
R1	25CC
R2	F012

INDIRIZZO	2000
PAROLA DI STATO	0045
Vettore di interruzione	

Origine dello stack del nucleo	0FFF
--------------------------------	------



Ipotesi: due banchi di registri (registri generali + SP)

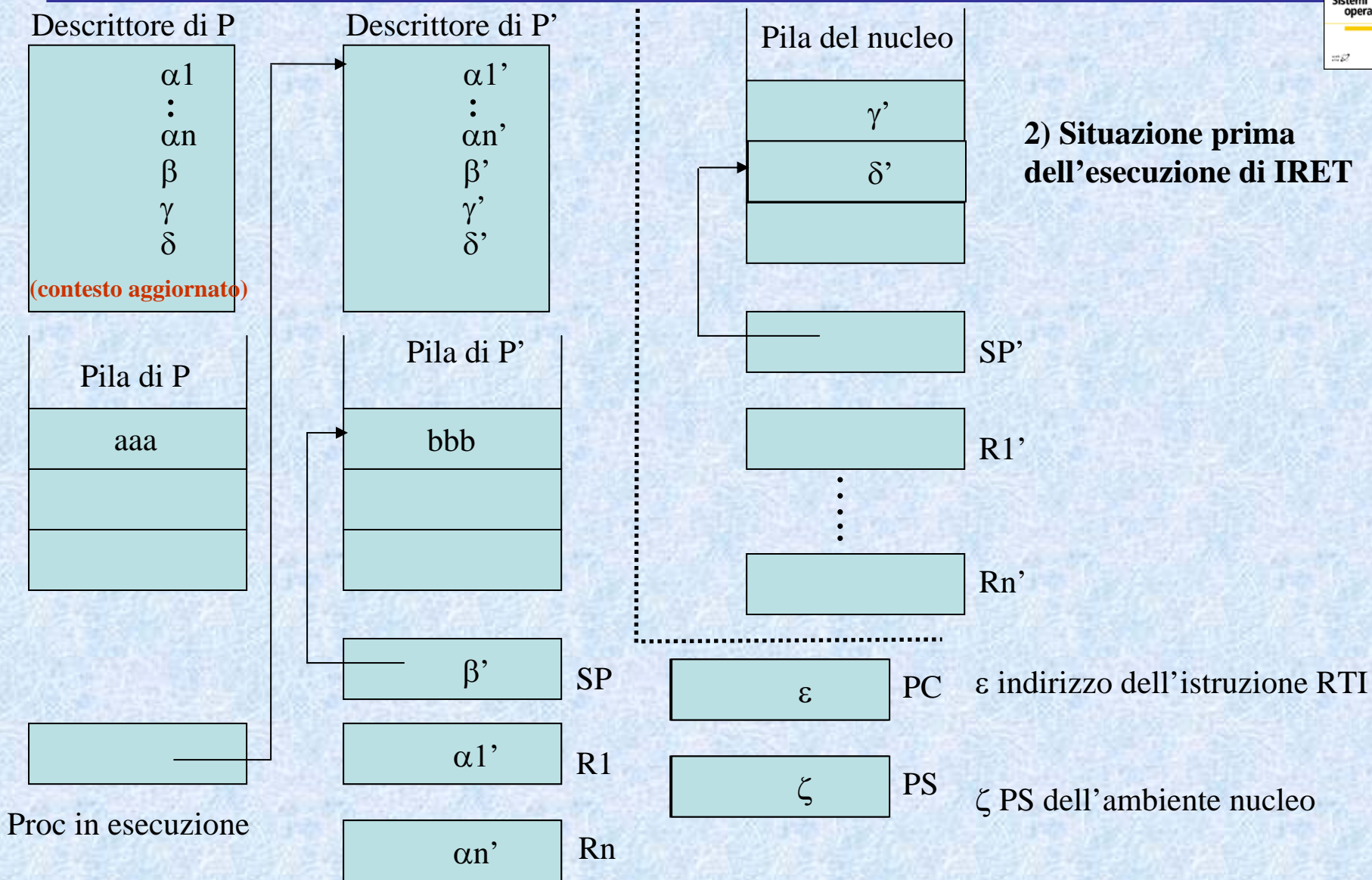
1) Situazione dopo l'esecuzione della SVC

β : SP del processo che ha eseguito SVC; γ : PC del processo che ha eseguito SVC

δ : PS del processo che ha eseguito SVC

ε : indirizzo della funzione di servizio

ζ : PS dell'ambiente nucleo



2.5 Scheduling dei processi

Attività mediante la quale il sistema operativo effettua scelte tra i processi,

principalmente riguardo a:

- assegnazione del processore
- revoca del processore

=>> scheduling a breve termine

ma anche (eventualmente):

- prima inclusione nell'insieme dei processi schedulabili
=>> scheduling a lungo termine
- temporanea esclusione da e nuovo inserimento in questo insieme

=>> scheduling a medio termine

Tre livelli di scheduling:

– Scheduling a breve termine

- Sceglie tra i processi pronti quello a cui assegnare il processore
 - non preemptive scheduling
 - preemptive scheduling*(preemption = prelazione, prerilascio, revoca)*

– Scheduling a medio termine (Swapping)

- trasferimento temporaneo in memoria secondaria di processi
 - quando la disponibilità di memoria principale è inferiore alla necessità dei processi esistenti
- (= esclusione/reinserimento nell'insieme dei processi schedulabili)*

– Scheduling a lungo termine

- Sceglie tra i lavori in arrivo quelli da ammettere per la prima volta tra i processi schedulabili
 - controlla il grado di multiprogrammazione
- (tipico dei sistemi batch multiprogrammati)*

2.6 Operazioni sui processi

- **Meccanismi per la gestione dei processi:**
 - Generazione
 - Terminazione
 - Interazione
- ==> chiamate di sistema**

Generazione di processi

- Processi permanenti (creati)
- Processi temporanei (generati)

Meccanismi per la generazione dei processi:

- Un processo (padre) può richiedere la generazione di nuovi processi (figli)
- *mediante una chiamata di sistema*

esempio: nei sistemi interattivi il processo *Login* genera processi figli per l'esecuzione di programmi applicativi

=>> gerarchie di processi

=>> dopo la generazione, il processo padre può passare in stato di attesa della terminazione del figlio, oppure evolversi concorrentemente ad esso

Terminazione di processi

Un processo termina :

- Quando esegue una *chiamata di sistema* per la propria terminazione
- Quando il S.O. rileva un evento eccezionale

Tipicamente:

- esecuzione di un'operazione illecita: ad esempio il tentativo di accedere alla memoria privata di altri processi
(segnalato da un'eccezione)
- errore che non permette di continuare l'esecuzione del programma (esempio: overflow, etc)
(segnalato da un'eccezione)

==> La chiamata di sistema o l'eccezione attivano lo scheduling dei processi

Terminazione di processi

- Ogni processo temporaneo:
 - è figlio di un altro processo
 - può essere a sua volta padre di altri processi
- Il sistema operativo mantiene le informazioni relative alle relazioni parentali (nel descrittore)
- Se un processo termina:
 - in generale deve essere rilevato il suo stato di terminazione
UNIX: lo fa il padre con una chiamata di sistema
 - Eccezionalmente: il padre può terminare prima della terminazione di un figlio
UNIX: in questo caso chi rileva lo stato di terminazione dei figli?

2.6 Modello a processi

- Ambiente globale

i processi possono condividere dati

==> ammessa “memoria comune”

- Ambiente locale

i processi non possono condividere dati

==> non ammessa “memoria comune”

Modello a processi con ambiente globale

Nel modello con ambiente globale, gli spazi di indirizzamento di due o più processi hanno un'intersezione non vuota, che contiene *i dati condivisi*.

==> La globalità dell'ambiente si riferisce ai dati, non al codice

- **Le interazioni tra processi avvengono attraverso i dati condivisi**

==> I valori assegnati da P_i a un dato condiviso possono essere letti da P_j , e viceversa

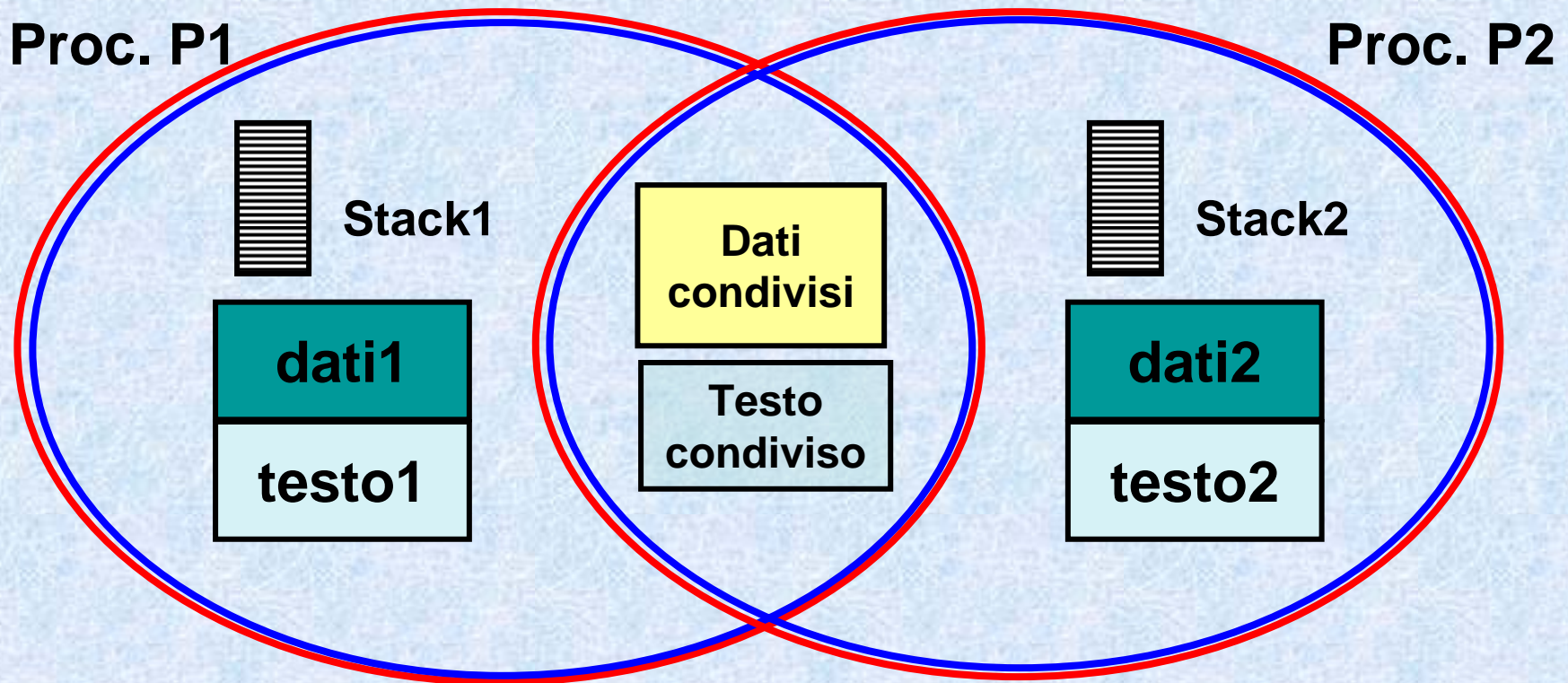
- **Le interazioni sono molto semplici, ma poco controllate**

==> Possibilità di interferenze

==> Necessari meccanismi di mutua esclusione

Modello ad ambiente globale

- 1) **Spazi di indirizzamento (logici)** e **aree di memoria (fisiche)** hanno intersezione non vuota



Modello a processi con ambiente locale

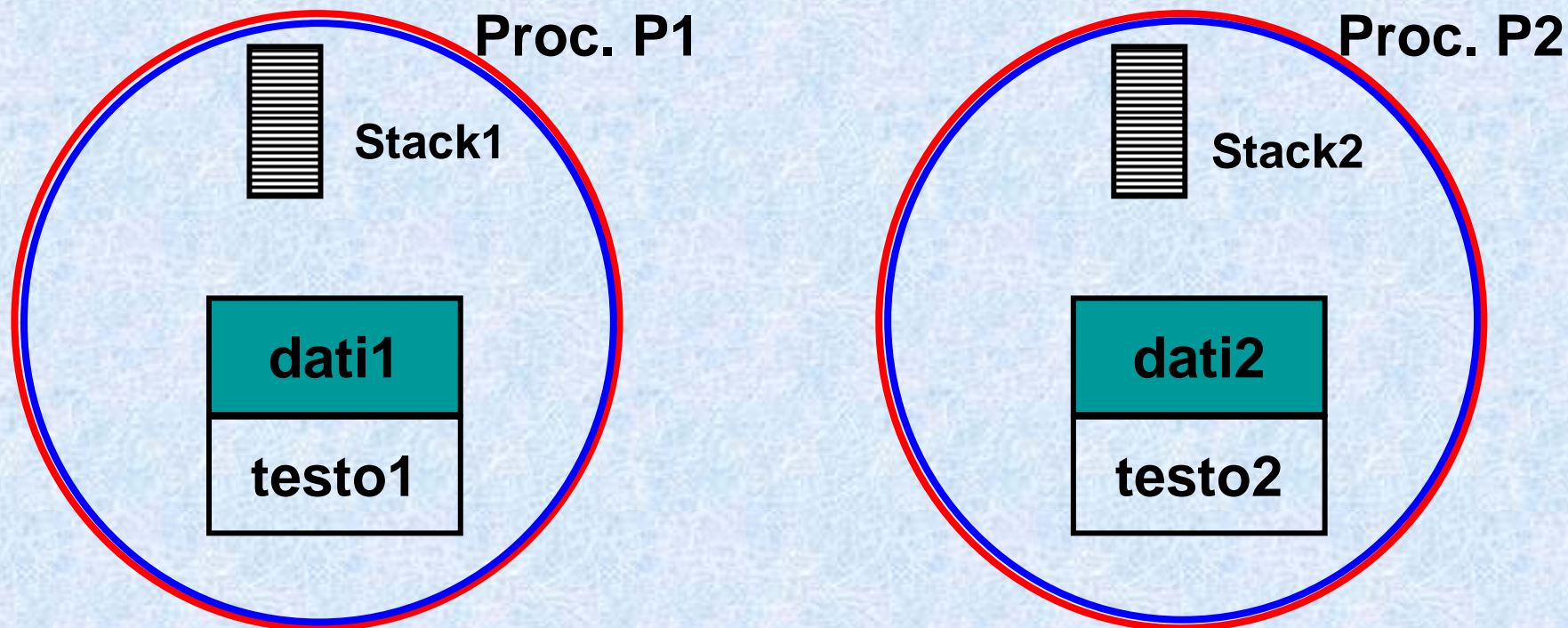
- Nel modello con ambiente locale, ogni processo ha uno spazio di indirizzamento privato *per i dati*.
==> L'ambiente locale ammette condivisione del codice. Esempio: i processi controllati da uno stesso programma condividono tutto il codice
- Le interazioni tra processi avvengono con meccanismi di IPC (*inter-process communication*), messi a disposizione dal kernel
 - esempio : *send, receive,*

Questo implica alti costi di interazione:

- *costo delle primitive di comunicazione*
- *costo delle commutazioni di contesto indotte*

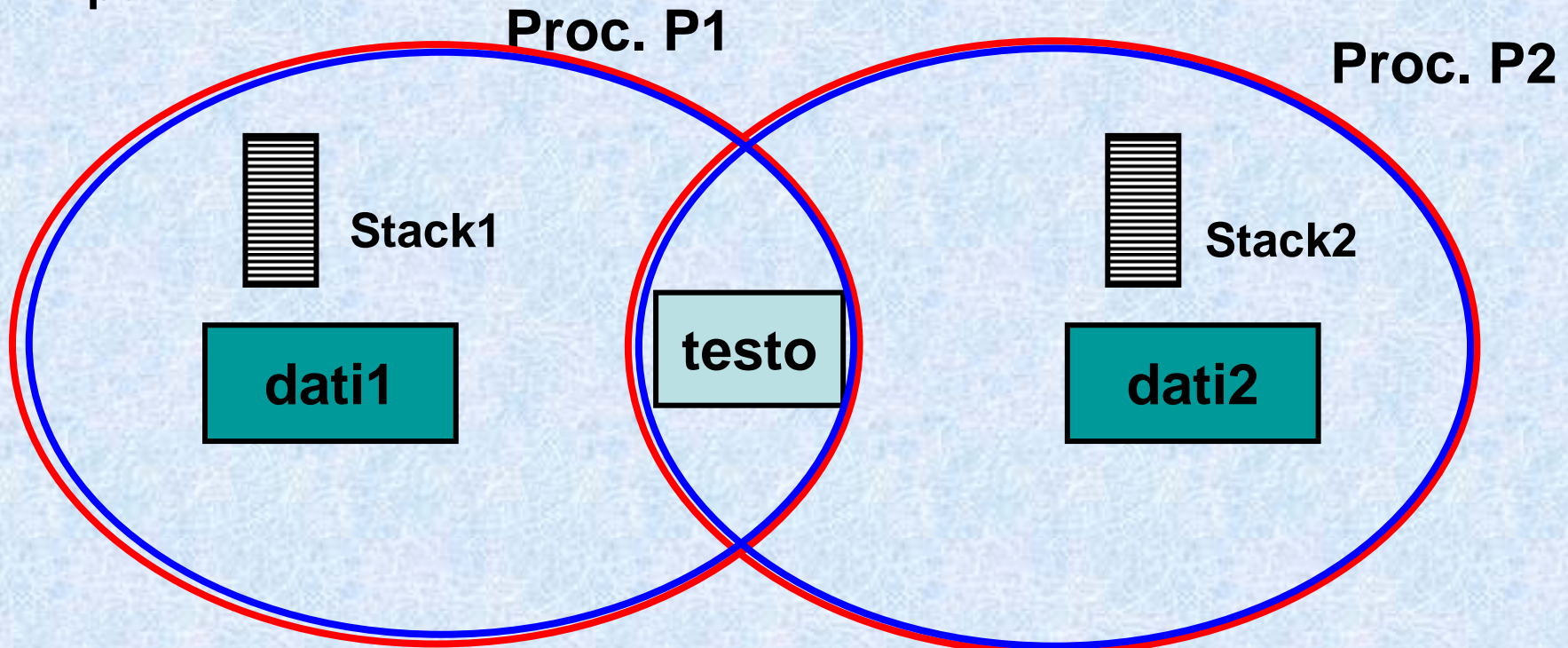
Modello ad ambiente locale

1) Spazi di indirizzamento (logici) e aree di memoria (fisiche) separati



Modello ad ambiente locale

2) Separazione degli **spazi logici** e delle **aree di memoria (fisiche)** solo per i dati



2.7 Flussi di esecuzione

Una stessa applicazione può contenere più flussi di esecuzione che avanzano con discontinuità

Esempio: flussi in un editor di testi

1. *Gestione del mouse*
2. *Immissione da tastiera*
3. *Gestione della struttura dati del testo*
4. *Visualizzazione sul terminale*
5. *Controllo ortografico*
6. *Salvataggio periodico nell'archivio*

La realizzazione dei diversi flussi mediante processi distinti comporta frequenti interazioni con conseguenti commutazioni di contesto.

==> per ogni processo, il tempo di permanenza in esecuzione potrebbe essere breve rispetto a quello necessario per la commutazione di contesto.

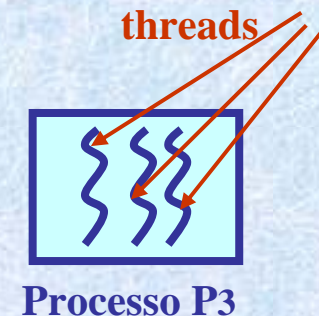
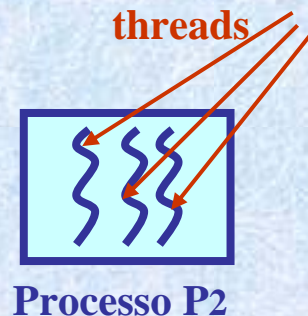
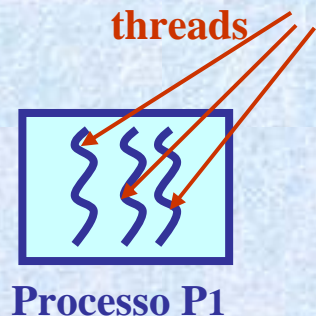
Esigenza: semplificare al massimo le commutazioni tra i flussi

Thread

Un thread realizza un flusso di esecuzione all'interno di un processo

- i thread di uno stesso processo condividono lo (o parte dello) spazio di indirizzamento del processo e le altre risorse*

Multithreading: il S.O. realizza flussi di esecuzione multipli all'interno dei singoli processi



Multithreading

Esempio: editore di testi con multithreading

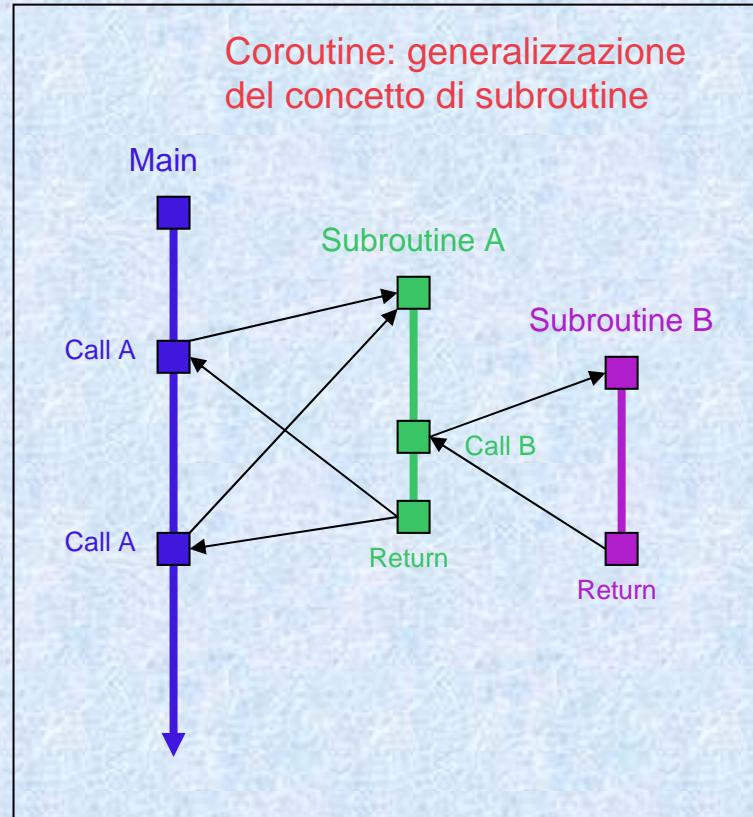
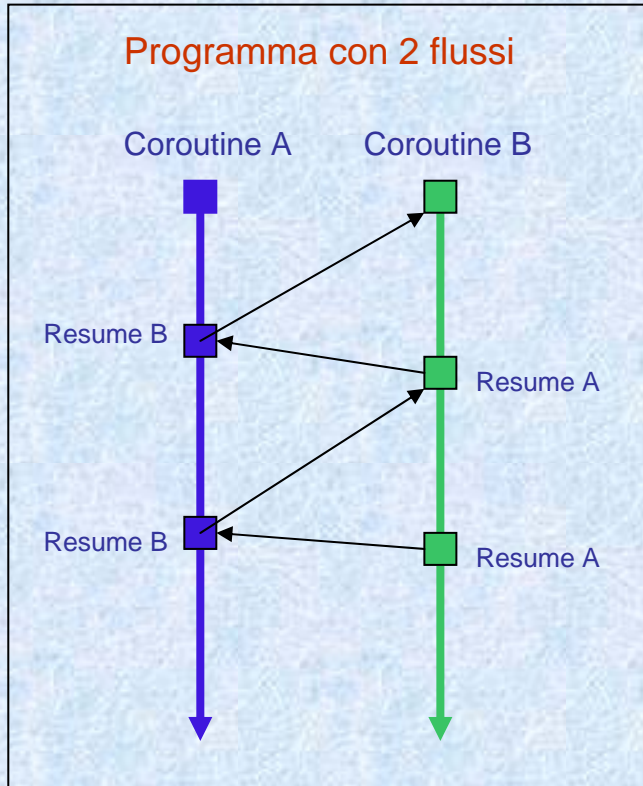
Processo: possiede le risorse

- Memoria
- Archivio (memoria secondaria)
- Dispositivi di I/O (tastiera, video, mouse)

Threads: corrispondono ai flussi di esecuzione

1. Gestione del mouse
2. Immissione da tastiera
3. Gestione della struttura dati del testo
4. Visualizzazione sul terminale
5. Controllo ortografico
6. Salvataggio periodico nell'archivio

Prima dei thread: le coroutines



Gestione dei flussi di un programma

- 1) Con le coroutines, l'alternanza in esecuzione dei flussi è a carico del programma
- 2) Con multithreading, l'alternanza in esecuzione dei flussi è:
 - a carico del nucleo del S.O. (thread "kernel level")
 - a carico di librerie dell'ambiente utente (thread "user level")

Thread

Threads di un processo:

- Condividono le risorse del processo
- Ogni thread ha stato, contesto e stack propri

Processo (*pesante*):

- Risorse di memoria
 - codice, dati
- altre risorse
 - Archivi aperti, ...
- Thread1, Thread2,

Thread (*processo leggero*):

- Condivide (*in tutto o in parte*) codice e dati del processo
- Possiede stato, contesto (PC, registri), stack, descrittore

Thread

Proprietà dei processi:

- Spazi di indirizzamento (generalmente) separati e reciprocamente protetti
- Interazioni che implicano chiamate di sistema e cambi di contesto
- Generazione e terminazione: operazioni complesse che implicano anche l'assegnazione e il rilascio di risorse

Proprietà dei thread:

- Condividono lo spazio di indirizzamento del processo cui appartengono
- Interazioni con modalità semplificate (attraverso dati comuni)
- Generazione e terminazione non implicano assegnazione e rilascio di risorse

Processi pesanti e processi leggeri

Processi senza multithreading:

- per ogni processo definito un unico flusso di esecuzione
- il processo è al tempo stesso:
 - un elemento del S.O che possiede risorse
 - unità di schedulazione del S.O

Processi con multithreading:

Separazione dei due aspetti:

- **Processo pesante (*task*):** elemento che possiede le risorse
- **Processo leggero (*thread*):** unità di schedulazione

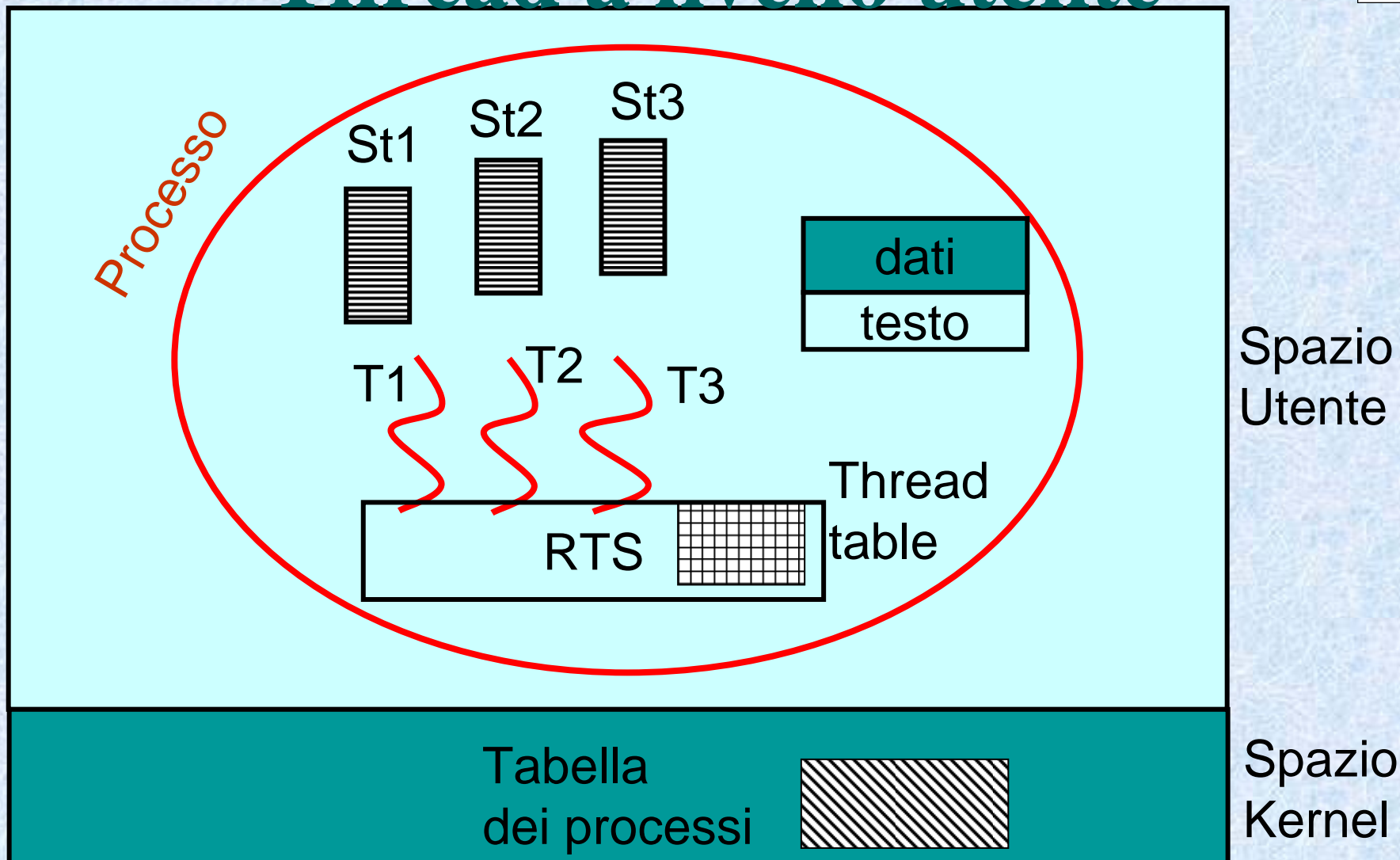
Thread a livello utente

Si utilizza un *Thread package*

(libreria realizzata in spazio utente)

- Gestione dei thread senza il supporto del S.O.
 - Lo scheduling dei thread è effettuato dal thread package (*supporto a run time dei processi utente*)
- Il S.O. si limita a gestire i processi

Thread a livello utente



Thread a livello utente

- Le Thread Table sono strutture private del processo
 - una thread table per ogni processo
- Scheduling dei thread realizzato nell'ambiente del processo
senza chiamate di sistema nè intervento del kernel
==> gestisce solo le commutazioni tra thread dello stesso processo
==> i thread possono rilasciare esplicitamente la CPU (*thread _yield*) per attivare lo scheduler dei thread
- Scheduling dei processi realizzato dal kernel
attivato da chiamate di sistema o da altre interruzioni
==> i thread non possono servirsi di chiamate di sistema per le proprie transizioni di stato e non possono ricevere direttamente le interruzioni: una chiamata di sistema bloccante eseguita da un thread (es., per attendere un servizio) provoca la sospensione del processo e conseguentemente di tutti i suoi thread
==> anche l'interruzione del timer per la gestione a quanti di tempo ha effetto sul processo e non sui singoli thread

Proprietà dei thread realizzati a livello utente

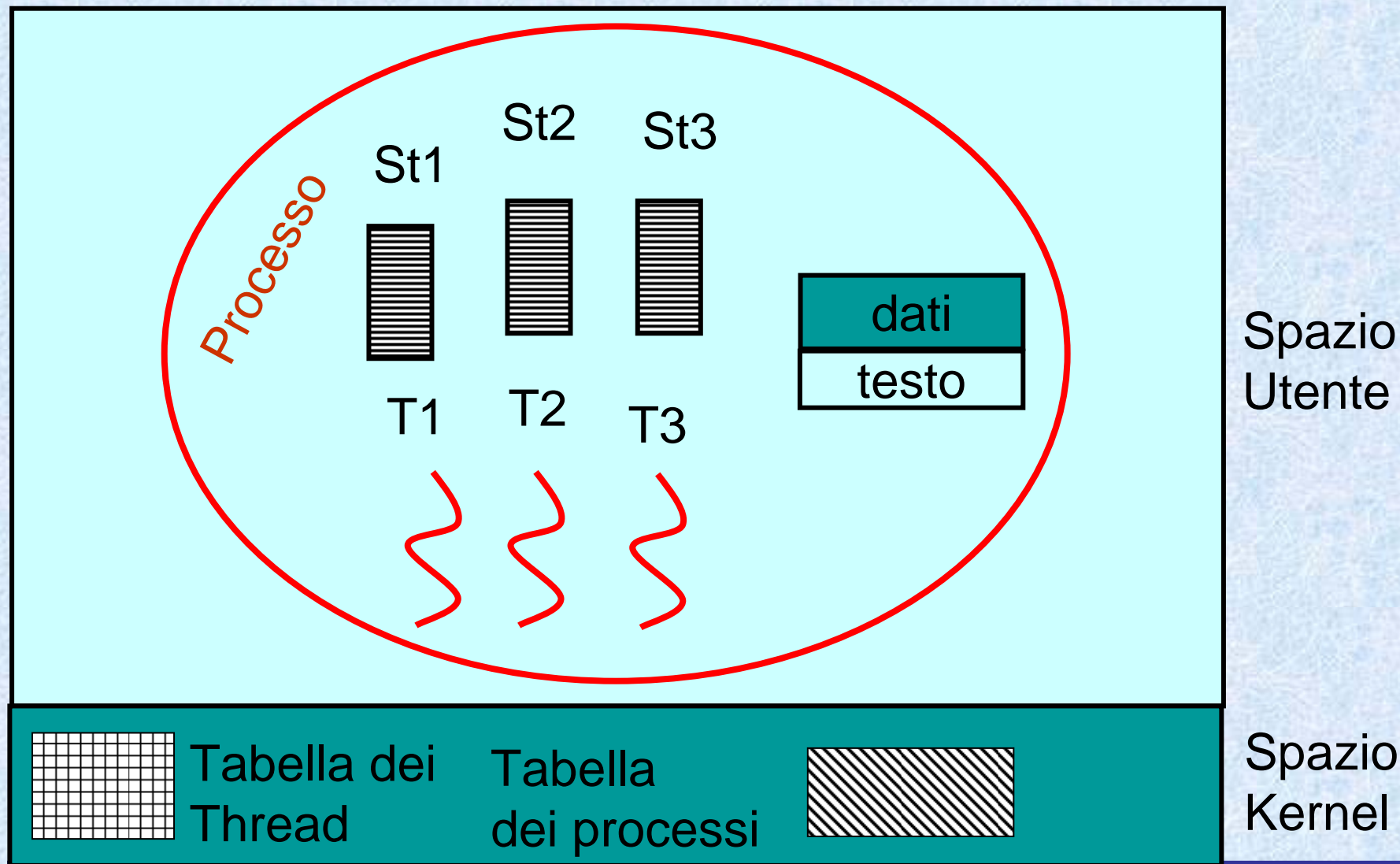
- Creazione, terminazione e commutazione di contesto molto veloce
 - *evitate le chiamate di sistema e la commutazione dello spazio di indirizzamento*
- possibilità di scheduling “personalizzato”, dipendente dall'applicazione
- realizzabili anche su un S.O. che supporti solo i processi
 - *esempio: molte versioni di UNIX*
- impossibilità di utilizzare chiamate di sistema bloccanti per le interazioni tra thread
 - ==> *necessario utilizzare meccanismi non bloccanti*
 - ==> *le interazioni possono essere inefficienti*
- **Thread a livello utente non traggono alcun vantaggio dalla disponibilità di sistemi multiprocessore**

Thread a livello del nucleo

- Thread completamente gestiti dal kernel
 - *generazione, terminazione, interazioni, schedulazione: attivate da chiamate di sistema*
- In molti casi i thread costituiscono le sole unità schedulabili
 - *gestite commutazioni tra thread qualsiasi (anche di processi diversi)*
 - *esempio: Windows, Linux*

==> Processi: esclusivamente contenitori di risorse (memoria, archivi aperti, ...)
- **Thread a livello kernel realizzabili anche nei sistemi multiprocessore**

Thread a livello del nucleo



Thread a livello kernel

- Thread table unica (nel kernel)
- Le operazioni sui thread e le reciproche interazioni sono realizzate con chiamate di sistema
 - *thread_create(), thread_exit(), thread_wait()...*
 - *in generale, i thread non rilasciano esplicitamente il processore***==> le chiamate di sistema possono bloccare un thread senza bloccare tutto il processo**
- Operazioni e interazioni comportano overhead maggiore rispetto al caso dei thread a livello utente
 - *costo delle chiamate di sistema*
 - *commutazione dello spazio di indirizzamento se la commutazione è tra thread di processi diversi*
 - *nella commutazione tra thread di uno stesso processo, evitabile l'invalidazione delle cache*